This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2016.2604923, IEEE Transactions on Computational Intelligence and AI in Games

Accelerating board games through Hardware/Software Co-Design

J. Olivito, J. Resano, and J.L. Briz

Abstract—Board games applications usually offer a great user experience when running on desktop computers. Powerful high performance processors working without energy restrictions successfully deal with the exploration of large game trees, delivering strong play to satisfy demanding users. However, nowadays, more and more game players are running these games on smartphones and tablets, where the lower computational power and limited power budget yield a much weaker play.

Recent Systems-on-a-Chip include programmable logic tightly coupled with general-purpose processors enabling the inclusion of custom accelerators for any application to improve both performance and energy efficiency. In this article, we analyze the benefits of partitioning the artificial intelligence of board games into software and hardware. We have chosen as case studies three popular and complex board games, Reversi, Blokus, and Connect-6. The designs analyzed include hardware accelerators for board processing which improve performance and energy efficiency by an order of magnitude leading to much stronger and battery-aware applications.

The results demonstrate that the use of hardware/software co-design to develop board games allows sustaining or even improving the user experience across platforms while keeping power and energy low.

I. Introduction

BOARD games have been the target of many research efforts in the last decades. These works frequently present software implementations that are executed in desktop or server computers. However, currently, users prefer to play board games on mobile devices such as smartphones or tablets. The question is: Are the solutions developed for desktop computers directly applicable to mobile devices? The fact is that there is a big gap between the strength of the board game applications developed for desktop computers, and those developed for mobile devices. At first glance it might be thought that mobile processors do not provide enough performance. However, current Systems-on-a-Chip (SoC) developed for mobile devices include up to eight powerful out-of-order 64-bit cores running up to more than 2 GHz, providing a lot of computational power, which is more than enough to execute a strong board game player. The actual limit is the power budget. SoCs for mobile devices provide high peak performance, but running a computationally-intensive application drains the battery very fast, leading to a bad user experience.

To overcome this problem for most computer games, mobile SoCs include specialized hardware resources such as Graphic Processing Units (GPUs), or fixed hardware accelerators (called *ASICs*, *Application-Specific Integrated Circuits*) for frequently demanded functionality as decoding high definition audio and video. These resources not only provide high performance but are also energy efficient compared with a general-purpose processor. However, the computational complexity of many board games is not due to graphics, video, or audio processing, and hence they cannot take advantage of these hardware resources. Moreover, adding specialized fixed accelerators for board games is not a feasible solution, since each board game has different demands, and not all the users need additional support for these applications. Fortunately, there is another option: exploiting the programmable logic included in recent SoCs, which combine low-power processors and *Field Programmable Gate Arrays (FPGAs)*

FPGAs are nowadays the most broadly used programmable logic devices. They constitute a mature technology that has been proved to greatly increase performance and reduce energy consumption on many different applications [1-6]. Board games are excellent candidates to make use of this technology since the computations involved in solving these games exhibit a large degree of fine-grained parallelism. Moreover, FPGAs are flexible and reusable. They can virtually implement any hardware logic by loading the proper configuration (i.e. programming the FPGA), and its functionality can be changed as many times as needed, even at run-time. Hence the same hardware resources can be used to provide hardware acceleration for different applications.

The main FPGA manufacturers, Xilinx and Altera, have released complete processor-based *System-on-a-Chip* (SoCs) with an FPGA integrated in a single chip (Zynq-7000 SoC and Zynq UltraScale+ MPSoc by Xilinx [7], and Arria V [8], and Stratix 10 by Altera [9]). These platforms are similar to those found in mobile devices, but including FPGA resources on chip tightly integrated with the low-power processors. They allow software developers to use known programming environments, while logic designers can use the FPGA to introduce customized features to improve performance and reduce energy consumption. Moreover, leading manufacturers like Intel, IBM, and Qualcomm have recently announced that they are preparing SoCs including processors and FPGAs. Other companies such as Menta and Flex Logic have designed their own Intellectual Property (IP) FPGA core, which can be included in any SoC at a reduced cost. Hence, FPGAs are expected to be frequently found in mobile SoCs in the near future, just as GPUs are nowadays.

Coding hardware for FPGAs involves an additional development effort. Although both Xilinx and Altera provide High-Level Synthesis (HLS) tools to simplify the process of

mapping a software solution to an FPGA [10], this is still not straightforward. Hardware/software co-design greatly mitigates this effort by keeping most of the functionality in software, and moving to hardware only computationally intensive data-processing cores.

In this article we propose a co-design approach for board games. The idea is to code the control of the artificial intelligence in software and move board-processing computations to hardware accelerators. In other words, the hardware will process the boards in order to extract all the useful information, and the software will use that information to follow any given strategy to explore the search space. Board processing cannot be efficiently parallelized in general-purpose cores because the size of the board is not big enough to compensate for the parallelization overhead. Moreover, it is not suitable to leverage the Single-Instruction Multiple-Data (SIMD) units included in modern processors either. These units execute the same arithmetic instructions on different data, but each board position in a board game may demand a different computational treatment. On the contrary, a custom Multiple-Instruction Multiple-Data (MIMD) unit implemented on an FPGA can perfectly face this problem.

We carried out our analysis on three complex board games: Reversi, Blokus Duo, and Connect6. These games were selected by the design competition committee of the International Conference on Field Programmable Technology. We were awarded the first prize in two of these competitions and the second prize in the other one, and thanks to these experiences we get insight about these games. We first developed and optimized a full software application for each game, and then we included hardware accelerators to process boards faster. The techniques implemented to explore the search space are minimax with alpha-beta pruning, iterative deepening, and full node ordering on the first two tree levels according to the previous shallower search. We selected these techniques because they are frequently found in board games, and they are enough to build a proof of concept. It is important to remark that we focus on board processing, and our co-design approach allows modifying the exploration techniques without needing to redesign the hardware accelerators. Tasks such as finding the legal moves or evaluating a board are always needed, regardless of the techniques selected to explore the search space. The software and hardware has been designed targeting the Xilinx Zynq 7000 SoC, which includes a dual-core ARM Cortex-A9 processor interconnected with an FPGA in a single chip. As an additional reference, we also evaluated bare full software versions, executed on a high-performance Intel i7-2600 processor.

The objective of this article is to evaluate the potential of hardware/software co-design to develop stronger AI engines for board games, especially in battery-dependent systems where the computational power and energy budget are limited. With current co-design environments, it is possible to design hybrid processor/FPGA systems where the FPGA can be used to speed up the most critical computations leading to better performance and less energy consumption with a reasonable development effort, which is desirable in high-performance mobile computing. The results demonstrate that splitting the board games applications into hardware and software parts, allows the designers either to develop stronger opponents, or to reduce the energy consumption, while keeping reasonable development cycles.

II. Related Work

Board games, especially Chess, attract the interest of the community not only because of their popularity but because they pose the challenge of developing computer players strong enough to beat the best human players. Deep Blue reached the most memorable milestone in 1997 when it was able to defeat the world champion at that time, Gary Kasparov. Hardware accelerators played a key role to succeed [11]. These accelerators were *Application Specific Integrated Circuits* (*ASICs*) specifically designed for that system. ASICs design provides the best performance and energy efficiency balance, but it also involves large development cycles and in most cases unaffordable costs [12].

The emergence of programmable logic turned the design of custom hardware into a feasible option, dramatically shortening the development cycle and lowering costs. Several works have described implementations of board games in FPGAs. In [13] Wong *et al.* presented an implementation on the Reversi game. Their design reached a 3.67 speedup over an equivalent software running on a high-end processor. Later in 2014, Olivito *et al.* elaborated a comprehensive comparison of hardware and software implementations of Reversi in terms of performance and power, and pointed out that the hardware implementation on a low-cost FPGA was able to perform 25 times faster while consuming 400 times less power than the software implementation running on a high-end processor [14]. Other games like Connect6, Blokus and Go have also been implemented by the FPGA developer's community. The works presented in [15-17] detail FPGA-based implementations and comparisons with software, reporting speedups of one or even two orders of magnitude. In the light of these results, it is clear that FPGAs outperform general purpose processors in these games. However, the design and implementation of the whole artificial intelligence purely in hardware requires a much larger development cycle than an equivalent software design, preventing the use of this technology.

Hardware/Software co-design combines the flexibility and short development cycles of software design with the higher performance and lower power consumption of FPGAs. Early co-designs were based on systems where the CPU and the FPGA were in different chips, communicated through a system bus. One of the first applications of co-design to accelerate board games was published in 2002 [18]. This work presents a Chess player in which the move generation was accelerated by an FPGA and the remaining tasks of the AI were executed on a processor. In 2004, another successful use of processors and FPGAs to accelerate a Chess program was presented in [19]. Brutus was one of the strongest chess programs at that time and one of its key design strategies was to split the tree search into software and hardware. In these previous approaches the CPU/FPGA communication overhead was a limiting factor both for the granularity and the speedups obtained by the tasks moved to hardware. The new

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2016.2604923, IEEE Transactions on Computational Intelligence and AI in Games

heterogeneous SoCs, which integrate processors and FPGAs in the same chip, take weight off this issue. Moreover, manufacturers provide co-design environments with tools that automatically generate bus interfaces. Hence, communications are not only more efficient but also easier to manage.

III. Hardware/Software Co-Design

Hardware/software co-design allows designers to partition an application into hardware and software blocks that interact among them. Profiling the application in order to identify which tasks demand hardware acceleration, and reducing the communication overheads are the keys to develop a good co-design solution.

A. Zynq Processor/FPGA Platform

Zynq-7000 is a SoC which integrates a dual-core ARM Cortex-A9 general-purpose processor, and an FPGA in a single chip. This heterogeneous platform joins up software flexibility and hardware efficiency, allowing developers to differentiate their products by increasing performance and energy efficiency. A critical aspect for hardware/software co-design to succeed is to enable an efficient communication between the processor and the programmable logic. The speedup achieved by the custom hardware must compensate for the communication overhead.

The communication between the ARM processor and the FPGA in the Xilinx Zynq devices is performed through an AXI4 interconnection bus [20]. It facilitates IP integration saving development time while providing high throughput and low latency. This bus offers several configurations, optimized to different traffic profiles. Our design leverages AXI-Lite and AXI-Stream. AXI-Lite is suitable for small transfers. With this interface the hardware accelerator is assigned a set of 32-bit registers mapped into the processor memory space. Communicating hardware and software is as simple as writing or reading these registers. On the other hand, AXI-Stream is suitable for large transfers thanks to their burst mode. When using this interface, a DMA sends the data back and forth through the AXI ACP (Accelerator Coherency Port) which ensures cache coherency when a hardware module modifies the memory without processor intervention.

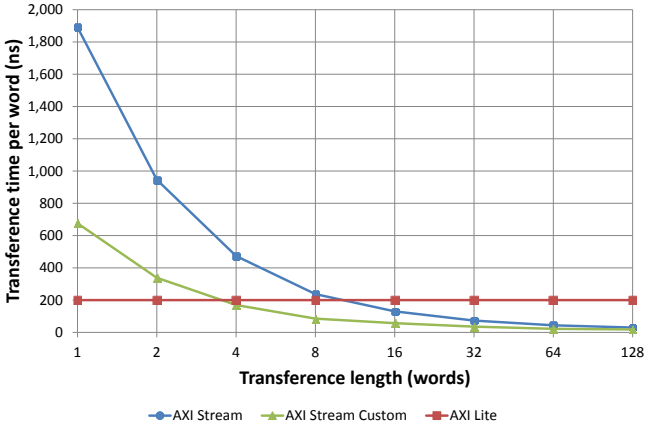


Fig. 1. Transference throughput of each AXI interface

As a first step in our co-design analysis, we have measured the communication latency of these two options for different transfer lengths, since this information is critical to develop an efficient communication scheme. As it can be seen in Fig. 1, the throughput of the AXI-Lite interface is constant because each transfer always sends a single word. Instead, the time in AXI-Stream burst-based transfers decreases logarithmically as the transfer size increases. AXI-Stream Custom uses the same hardware that AXI-Stream but simplifies the driver by assuming that the source and destination addresses are always the same along the execution of an application. This assumption is valid in our applications and greatly increases the throughput for medium sized transfers.

IV. Methodology

For each case study, we developed the game application entirely in software. They were written in C and compiled with GCC 4.9.2 with the maximum optimization flag activated; then we gathered data to select the kernels to accelerate. To this end we used Intel VTune Amplifier XE 2016 running the games on an Intel i7-2600 processor. Intel VTune leverages dedicated hardware counters on Intel processors to perform a non-intrusive and statistical profiling.

Once the hotspots were identified, we first tried to parallelize the software versions using different thread libraries (POSIX Threads and Intel Threading Building Blocks), or the powerful SIMD extensions, but neither of these options improved the results. Then we developed hardware modules to accelerate those bottlenecks. These hardware accelerators were written in VHDL and synthetized with the toolchain of Xilinx Vivado 2015.2. The ARM and FPGA communication is fully assisted by the tools, being all the AXI interfaces and the DMA controller self-generated modules.

Power consumption measurements were taken with a Yokogawa WT210 digital power meter, a device accepted by Standard Performance Evaluation Corporation (SPEC). We are interested in the power consumed due to the execution of our applications, so we measured the power consumed both in an idle state and while executing our applications, and considered the difference on average.

V. Case Study I: Reversi

Reversi is a strategy board game played between two players on an 8 x 8 board with discs colored black on one side and white on the opposite side. Each player shall be assigned to play a color. The goal of the game is to have more discs than the opponent at the end of the game.

The game will start with blacks making a move. The play then alternates between whites and blacks until one of the two following situations occurs: a) There are no moves that the player can make to outflank the opponent's disc(s) (the player is then said to have no valid moves) or b) both players have no valid moves.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2016.2604923, IEEE Transactions on Computational Intelligence and AI in Games

When a player has no valid moves, he forfeits his turn and the opponent continues to move. A player is not allowed to voluntarily forfeit his turn. The game ends when both players have no valid moves or when the entire board has been played. Therefore, it is possible for a game to end before all 64 squares are filled.

A. *Techniques implemented*

The board evaluation is based on strategic concepts such as mobility, which is the number of legal moves; stable discs, which are those discs cannot be flipped anymore; corners capture; and number of discs.

B. *Hardware acceleration*

The profiling revealed that 89.3% of the game time is invested in the evaluation of boards – 48.6% computing stable discs, and 40.1% computing the mobility -, 9.6% of the game time is spent in the move generation, and only 1.1% is due to the execution of the game-tree search algorithm. Hence, we developed hardware accelerators for the two tasks involved in board evaluation:

- Mobility: In order to compute the mobility of each player, every board square must be analyzed to determine whether it corresponds to a legal move or not, by checking different patterns in its row, column and diagonals. This task exhibits a great degree of data parallelism as hundreds of patterns must be checked. A hardware module can seamlessly exploit this parallelism since it is able to check all the patterns for all the squares at the same time. **Hence, all the legal moves are identified in just one clock cycle**. Fig. 2 points out the legal moves for the white player in this board and the hardware cell that checks all the patterns in parallel. A legal move must satisfy two conditions, that the square is empty, and that at least one opponent's disc is followed by an own disc in any direction. The hardware module that computes the mobility of a player consists of 60 cells like the one shown in the figure.

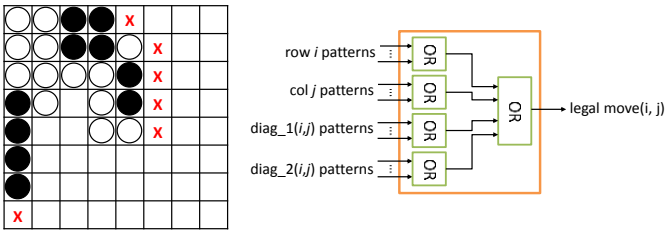


Fig. 2. Cell architecture to determine whether a square corresponds to a legal move.

- Stable discs: Another metric commonly used to evaluate boards in the Reversi game is the number of stable discs. A discs is stable if at least one of their two neighbors in each direction are stable. Fig. 3 marks in green the stable discs of the example board and illustrates the hardware cell that determines if a discs is stable.

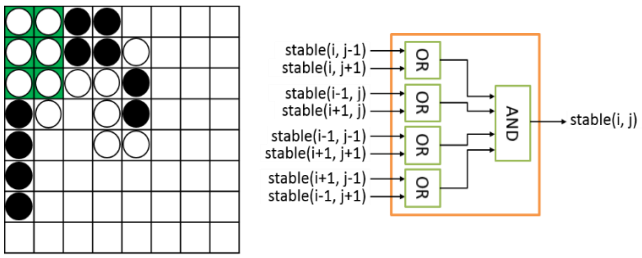


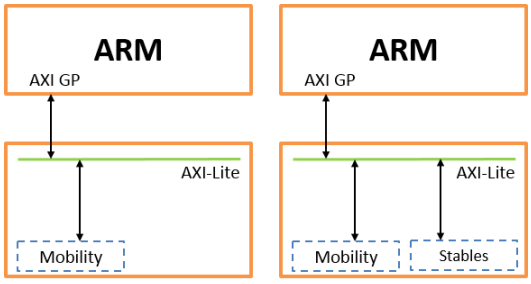
Fig. 3 Cell architecture to detect stable discs

C. *Co-design schemes*

We have implemented two co-design schemes for the Reversi game. The first one moves the computation of the metric mobility, which is one of the two hotspots, to the hardware side, and the other one moves the whole evaluation task – mobility and stable discs -. The input of the hardware accelerator in both cases is the board to evaluate, which is coded in software as an 8x8 matrix of 1-byte elements, but, in order to reduce the communication overhead, these data are compacted to only two bits per square by means of bitwise operations. We selected the AXI-Lite interface in all the schemes because it offers the lowest overhead for the required transfer sizes.

The output in the scheme (a) is the mobility of both players, which are values from 0 to 60, and therefore 12 bits are required to encode both values. The transfer wide in the AXI-Lite bus is 32 bits, so each board evaluation needs four transfers from the processor to the FPGA to send the board, and one from the FPGA to the processor to read back the mobility of both players.

The output in the scheme (b) includes also the number of stable discs of both players. The range of values is the same as in the mobility, so each board evaluation reads a total of 24 bits from the hardware accelerator, adding no overhead with regard to the first scheme, since the transfer size is 32-bits.



(a) (b)

Fig. 4. Co-design schemes for the Reversi application

VI. Case-Study II: Blokus

Blokus is an abstract strategy board game for two to four players, invented by Bernard Tavitian and first released in 2000. Blokus Duo is a variant of Blokus designed for only two players that use a smaller (14×14) board. This game is becoming increasingly popular because its rules are simple

and games become fast and dynamic.

Each player has a set of 21 different-shaped tiles, and can place them with eight different rotations. Each set (i.e. player) has a different color. The tiles can be placed only in those squares with corner-to-corner contact with a tile of the same color. Moreover, a new tile cannot have edge-to-edge contact with any other tile of the same color. Each player places one tile at one time, and the game continues until neither of them can place tiles anymore. The score of each player depends on the number of placed tiles and their size. The larger tiles (five squares) add five points, and the smaller one (one square) adds one point. Hence the objective is to occupy as many squares as you can with your tiles, while trying to reduce the number of squares available to your opponent.

A. *Techniques implemented*

Our Blokus application evaluates boards according to the metric *accessibility*, which quantifies the squares that are potentially reachable. A square is reachable if can be occupied by means of a legal move of the given player. A player with more *accessibility* during the game has more chances to win the game.

The computation of this metric is performed in two steps. The first one looks for the tiles' corners where a player can place a tile by satisfying the corner-to-corner rule. The second one analyzes the surroundings in order to check whether they are reachable or not. This step involves many pattern comparisons, which are amenable to be performed in parallel.

In addition, *accessibility* is also used to reduce the effective branch factor of the search tree by exploring only movements in areas which are also reachable by the opponent. To this end, the application uses a structure called *overlapping map* which is used as a filter to select moves that fight for areas accessible by both players.

B. *Hardware acceleration*

Profiling our Blokus software application showed that it spends 92.7% of the time evaluating boards, 5.3% finding legal moves and generating new nodes, and 1.9% generating overlapping maps. According to these results, we decided to move the evaluation to hardware. We also moved the generation of overlapping maps, despite not being one of the larger hotspots, because the hardware developed to evaluate nodes does also provide such maps.

The hardware module that computes the accessibility, processes the board vertex-by-vertex, checking in parallel all the patterns for all the squares surrounding the vertex. As a result, this module is able to process a board in as many cycles as vertices. Fig. 5 illustrates the architecture of accelerator submodule which checks the accessibility surrounding a vertex.

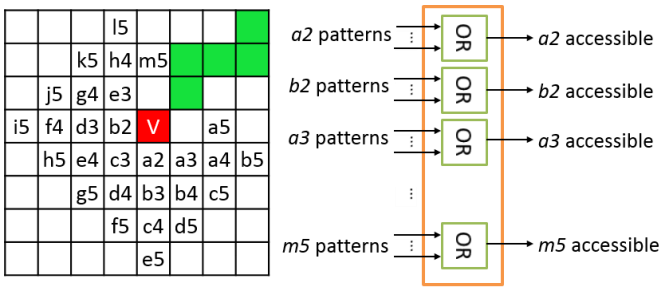
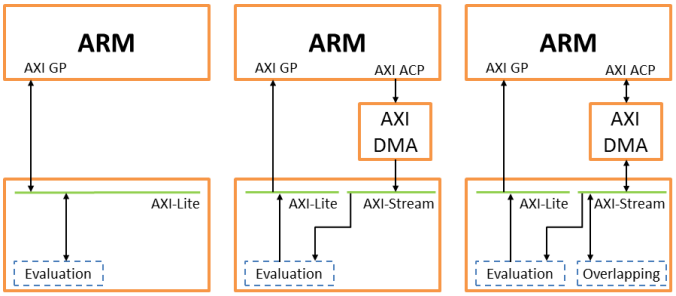


Fig. 5. Cell architecture to find the squares that are accessible from a given vertex

C. *Co-design schemes*

For this game we analyzed three co-design schemes. Schemes (a) and (b) have the same task distribution, the only difference is the way the board is sent from the processor to the accelerator. The size of the board in this game makes profitable the inclusion of a Direct Memory Access (DMA) in order to reduce the transference overhead according to the results presented in Section III.

Scheme (c) takes advantage of the evaluation hardware to compute the overlapping maps as well. Overlapping maps are sent from the accelerator to the processor through a DMA because of its size.



(a) (b) (c)

Fig. 6. Blokus co-design schemes

VII. Case Study III: Connect-6

Connect-6 is a board game that was introduced in 2003 by Professor I-Chen Wu. There are two players: black and white, each one playing with stones of the corresponding color. The game is played on a 19 x 19 Go board, and the stones are placed on the intersections. The black player moves first, placing one black stone on one intersection. Subsequently, white and black take turns, placing two stones on two different unoccupied spaces each turn. The first player that gets six or more stones of his color in a row (horizontally, vertically, or diagonally) wins.

A. *Techniques implemented*

We based the board evaluation in Connect-6 in the concept of *threat*. We name '*t4'* six contiguous squares with four discs of the same color and two empty squares, '*t3*' when containing three discs of the same color and three empty squares, and '*t2*' when two discs of the same color and four empty squares are found. Players shall try to make threats while defending from

them.

The analysis of the threats in a board requires to analyze every row, column, diagonal, and reverse diagonal. We name each of them '*section*', and we name '*window*' every possible combination of six contiguous squares within a section. Each section is analyzed following the algorithm detailed in [21]. This algorithm presents a data dependence since *t4* have to be analyzed in order to analyze *t3*, and *t3* in order to analyze *t2*. Fig. 7 show a trace of the steps that the algorithm follows to find the threats. In this example we first look for *t4*s and we find three windows that satisfy its definition. We select the leftmost one and place a mark in its rightmost empty square. We have identified one *t4* so far, and a new analysis reports another window with a *t4*. We mark it and the subsequent analysis does not find any *t4* anymore. The next analysis follows the same process looking for *t3*s, and finally the latest analysis will look for *t2*s. Note that, in software, each window within a section is traversed sequentially whereas in hardware all the windows can be processed in parallel.

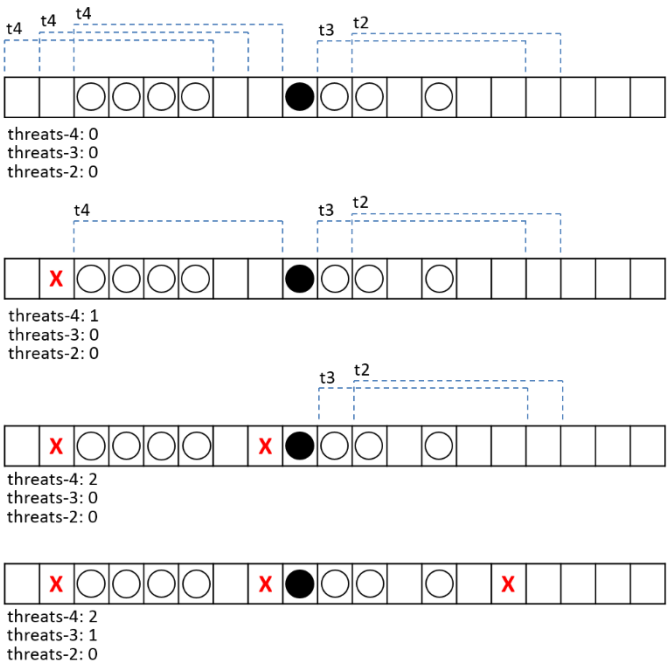


Fig. 7. Threat identification process

Threats are also used to steer the search-tree exploration. Positions that can upgrade to a threat are identified, and we explore first those corresponding to *t4*, then *t3* and finally *t2*. This approach is very similar to the scheme presented in [22].

B. Hardware accelerator

This application takes 90.4% of the execution time evaluating boards, 9.0% finding and selecting moves, and the remaining 0.6% is due to the min-max control. Fig. 8 details the architecture of the hardware accelerator developed to compute the threats in a board. *N* windows process in parallel the section under analysis, where *n* is sized to the number of windows that fit the section. *Next threat selector* is a priority encoder that updates the *marks* register with each new threat found, and increments the threat count of the current threat category.

The accelerator consists of one module per section in the board (19 rows, 19 columns, and 54 diagonals with at least 6 squares), and a tree adder to add the partial outcome of each section. This setup fully exploits the available data parallelism, being the section with the largest number of threats which determines the time required to fully compute a board.

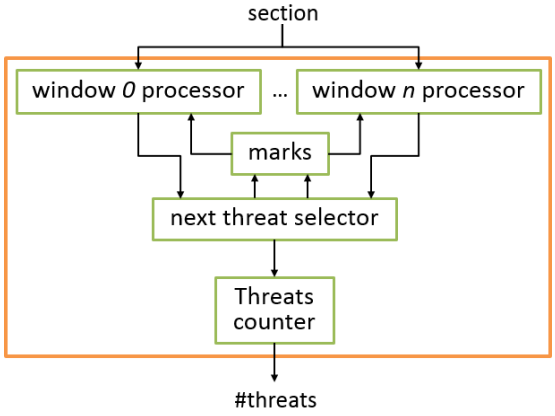


Fig. 8. Cell architecture to find threats in a section

C. Co-design schemes

In this case study, we only designed one co-design scheme. Board evaluation is clearly the target, taking more than 90% of the execution time. A hardware implementation of the task *next move selection* shares most of the hardware used to evaluate boards, but turns out quite complex and it is out of scope for this study.

The size of the board in this game is bigger than in the other case studies, making more profitable the use of the AXI-Stream interface to send the board from the processor to the accelerator. The evaluation value fits in a single word, and hence it is read from the accelerator through the AXI-Lite interface.

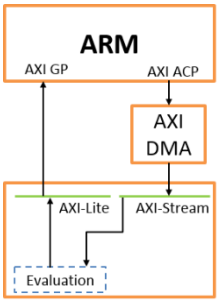


Fig. 9. Connect6 co-design scheme

VIII. Experimental Results

We implemented the software, hardware, and hybrid versions on the Xilinx Zynq platform (XC7Z020-CLG484 SoC) and then compared performance and power/energy

consumption. This platform includes an FPGA and an ARM dual-core Cortex A-9. As additional references, we also ran the software version on a desktop computer with an Intel i7-2600 processor, and we developed an additional co-design scheme where all the AI is implemented in the FPGA, and the ARM processor just manages the communications and the game procedure. We labeled this partition 'FPGA' because almost 100% of the computations were moved to hardware.

Tables I, II and III show experimental results for each case study. *Ex. Time* stands for the time required to complete a game; *Partitioning* details how the computation is distributed between the processor and the FPGA.; Δ*Power* represents the dynamic power consumption, i.e. the average increase in power consumption due to the execution of our application; and numbers in the column *Energy* are the product of power and execution time, which represents the energy consumed during a game due to the execution of our application.

We obtained these measures with the search tree exploring eight moves in advance in the case of the Reversi, four moves for the Blokus, and three moves in the case of the Connect6. With these parameters, our Reversi application explores 14.4 million boards during a game, the Blokus application explores 38.4 million, and the Connect6 explores 4 million.

Table I. Reversi experimental results

Platform	Ex. Time (Seconds)	Partitioning (CPU - FPGA)	ΔPower (Watts)	Energy (Joules)
Intel i7	33.0	100.0% - 0.0%	24.19	798.270
ARM	365.8	100.0% - 0.0%	0.10	36.580
Hybrid (a)	189.9	46.5% - 53.5%	0.08	15.192
Hybrid (b)	57.7	8.1% - 91.9%	0.08	4.616
FPGA	3.6	0.0% - 100.0%	0.02	0.072

The results in Table I show that the co-designed solutions offer remarkable speedups over the bare software version for the Reversi game. The first hybrid design achieves a 1.9 speedup by moving to the programmable logic fabric the computations responsible for the 53.5% of the original computation time. The second co-design solution reaches a speedup of 6.3 by moving the whole board evaluation to the accelerator. This version, based on a low-power processor, approaches the Intel i7 processor performance while consuming 173 times less energy.

Table II. Blokus experimental results

Platform	Ex. Time (Seconds)	Partitioning (CPU - FPGA)	ΔPower (Watts)	Energy (Joules)
Intel i7	852.0	100.0% - 0.0%	28.555	24,328.9
ARM	8,615.5	100.0% - 0.0%	0.104	1,067.1
Hybrid (a)	652.8	7.4% - 92.6%	0.104	67.9
Hybrid (b)	614.1	7.4% - 92.6%	0.100	61.4
Hybrid (b*)	573.8	7.4% - 92.6%	0.100	57.4
Hybrid (c)	475.8	5.4% - 94.6%	0.093	44.2
Hybrid (c*)	427.8	5.4% - 94.6%	0.093	39.8
FPGA	28.7	0.0% - 100.0%	0.032	0.9

The results for the Blokus shown in Table II are impressive since the hybrid designs even outperform the Intel i7. The reasons are that the portion of the computation moved to the programmable logic fabric is greater, and that the size of the data transferences benefits from the high throughput offered by the AXI-Stream interface.

Hybrid designs are from 13x to 20x faster than the bare software application running on the ARM processor. This improvement in performance leads to huge energy savings. Notice that co-designs (b) - (b*), and (c) - (c*) have the same task partitioning and the only difference among them is the use of a customized DMA driver.

Table III. Connect-6 experimental results

Platform	Ex. Time (Seconds)	Partitioning (CPU - FPGA)	ΔPower (Watts)	Energy (Joules)
Intel i7	96.0	100.0% - 0.0%	29.973	2787.21
ARM A9	1244.6	100.0% - 0.0%	0.103	128.19
Hybrid (a)	116.5	9.6% - 90.4%	0.099	11.53
Hybrid (a*)	112.0	9.6% - 90.4%	0.099	11.09
FPGA	9.1	0.0% - 100.0%	0.027	0.25

In the case of Connect-6, moving the board evaluation to the hardware reduces execution time and the energy consumed by a factor of 11. As in the Reversi game, this co-design alternative with a low-power processor almost reaches the performance of the high-performance Intel i7, but requiring 250 times less energy.

In the three games the scheme that includes the whole AI in the FPGA, clearly outperforms the Intel i7, and reduces the energy several orders of magnitude. However, as we will explain later, this solution involves a much higher development effort.

Table IV. PS/PL Communication overhead

Design	Communication overhead	Data transferred (MB)	Throughput (MB/s)
Reversi (a)	15.7%	596.8	20.0
Reversi (b)	55.5%	641.0	20.0
Blokus (a)	19.5%	2543.2	20.0
Blokus (b)	14.4%	2543.2	28.7
Blokus (b*)	8.4%	2543.2	52.8
Blokus (c)	18.8%	2702.5	30.2
Blokus (c*)	9.0%	2702.5	70.4
Connect-6 (a)	8.9%	386.2	37.0
Connect-6 (a*)	4.5%	386.2	73.6

Table IV quantifies the impact of the communication between the processor and the programmable logic in terms of performance for the hybrid schemes. We collected the data by measuring the time spent moving data among them. Software versions store the boards in a two-dimensional array of 1-byte elements, whereas in hardware each square requires only two or three bits, depending on the game, and they are stored in registers. At the time of sending the board from the ARM to the FPGA, we compact the data by bitwise operations in order to reduce the communication overhead. This overhead is included in the communication overhead presented in the table.

Our co-design schemes send the board from the processor to the accelerator every time the accelerator is used and then sends back the evaluation value. There are design alternatives which could reduce the communication overhead, such as storing and managing the boards in the accelerator and then sending chains of movements instead of boards, but this kind

of design decisions imply a compromise between efficiency and design complexity.

Another interesting data is the reconfiguration latency (i.e. the time needed to properly load the accelerator onto the FPGA). This delay depends on the size of the configuration to load. In this case, using partial reconfiguration that only modifies a specific region of the FPGA, it is possible to load the games accelerators in 7 up to 70 ms, depending on the game. Notice that this step is done only once, when the application is opened.

IX. Design Complexity

The previous results demonstrate the usefulness of FPGAs to improve energy efficiency in board games. However, there are also some drawbacks associated to FPGA hardware design.

Hardware Description Languages (HDL) like VHDL or Verilog allow writing a preliminary version in HDL code in time comparable to the development in C language. However, writing HDL code ready to be translated into an efficient hardware implementation requires a good command on digital logic design, computer architecture concepts, and parallel computing. FPGA vendors are doing a great effort to simplify the hardware design process. For instance, Xilinx has developed a C/C++ to HDL compiler that can directly map C/C++ code to an FPGA. These tools are promising, but they still have much room for improvement.

The second drawback comes from the fact that generating the configuration needed to program an FPGA consumes much more time than compiling software code. As it can be observed in Table V, the compilation time when including hardware is two or even three orders of magnitude greater than the bare software designs.

The last and more relevant drawback is debugging complexity. In large designs, the high degree of parallelism involved, with many hardware modules and signals working at the same time, makes debugging FPGA systems even harder than the already difficult software parallel debugging. Moreover, behavioral simulations are slow, and simulating a few seconds of real execution time takes several hours. Again, FPGA vendors are helping at this point with better and better tools, like powerful simulators or specific support to monitor some FPGA internal signals. Moreover, the software version is very helpful when debugging the hardware version, since the data generated by both versions can be compare in order to identify the bugs.

Table V. FPGA resources utilization and development effort

Design	LUTs (%)	FFs (%)	BRAMs (%)	Compilation time (s)	**Design time (weeks)**
Reversi ARM	-	-	-	1.7	**7**
Reversi hybrid (a)	4.8	1.1	0.0	344.0	**8**
Reversi hybrid (b)	7.4	1.2	0.0	422.0	**8**
Reversi FPGA	10.4	0.9	2.9	538.0	**17**
Blokus ARM	-	-	-	2.9	**10**
Blokus hybrid (a)	29.3	1.9	0.0	911.0	**13**
Blokus hybrid (b)	29.7	4.9	1.4	1139.0	**13**
Blokus hybrid (c)	30.0	5.0	1.4	1247.0	**13**
Blokus FPGA	48.7	5.7	69.6	1771.0	**26**
Connect-6 ARM	-	-	-	2.4	**8**
Connect-6 hybrid	72.0	6.4	0.7	1545.0	**9**
Connect-6 FPGA	81.4	8.0	2.2	1953.2	**21**

The development of the software versions used in this work, including optimization and profiling, took from seven to ten weeks for each game.

Regarding the hybrid versions, the development and debugging of the hardware accelerator for the Reversi required only four days due to its simplicity. The accelerators for the Connect6 and Blokus are more complex, especially in the case of Blokus, and their development took one week and three weeks respectively. It is important to mention that the available design tools for co-design (in our case we use Xilinx Vivado [23]) are definitely helpful since the communication infrastructure and the hardware/software interfaces are generated automatically, and the software can interact with the hardware accelerator just as it is done with any other peripheral.

Finally, the development of the 'FPGA' versions took from four to six months for each game. Writing the code for these versions was not too complex, but debugging a hardware system that explores millions of boards was very challenging. We needed to include debugging support to identify the bugs, and perform time-consuming simulations.

X. Conclusions

Board games applications designed for the mobile market lack of strong AIs engines in most cases due both to the lower computational power and the energy restrictions of battery-dependent devices. Recent *SoCs*, which include programmable logic tightly integrated with the processor, allow the developers to include specific accelerators to process boards much faster, and therefore to deliver a stronger play on a power budget.

We have developed software and hybrid versions of three popular cross-platform games, and we have run them on a Zynq hybrid FPGA/processor platform. The results demonstrate that including accelerators on the FPGA to process boards increases the AI strength by drastically improving performance and reducing energy consumption.

In spite of the fact that development on FPGAs adds some complexity to the design process, hybrid hardware/software platforms pays-off the harder development cycle since non-critical tasks remain executed in the general-purpose processor, and the FPGA is reserved for specific and demanding tasks.

REFERENCES

[1] Lopez, S. et al., "The Promise of Reconfigurable Computing for Hyperspectral Imaging Onboard Systems: A Review and Trends," Proceedings of the IEEE, vol.101, no.3, pp.698-722, 2013.

[2] Cope, B. et al., "Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study," IEEE Transactions on Computers, vol. 59, no. 4, pp. 433–448, 2010.

[3] Seunghun, Jin et al. "FPGA design and implementation of a real-time stereo vision system." Circuits and Systems for Video Technology, IEEE Transactions on 20.1 (2010): 15-26.

[4] Cong, J. et al., "Customizable Domain-Specific Computing," Design & Test of Computers, IEEE , vol.28, no.2, pp.6,15, 2011.

[5] Lindtjorn, O., et al. "Beyond traditional microprocessors for geoscience high-performance computing applications." IEEE Micro 31.2 (2011): 41-49.

[6] Chelton, W.N. and Benaissa, M. "Elliptic curve cryptography on FPGA". Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2008, vol. 16, no 2, p. 198-205.

[7] Zynq-7000 SoC & Zynq UltraScale+ MPSoc http://www.xilinx.com/products/silicon-devices/soc.html

[8] Arria V SoC FPGA Hard Processor System http://www.altera.com/devices/fpga/arria-fpgas/arria-v/hard-processor-system/arrv-soc-hps.html

[9] Stratix 10 FPGA and SoC https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html

[10] Xilinx Vivado High-Level Synthesis http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[11] F. Hsu, "Chess hardware in Deep Blue", Computing in Science and Engineering, vol. 8, no. 1, pp. 50-60, 2006

[12] F. Hsu, "A Two-Million Moves/s CMOS Single-Chip Chess Move Generator" IEEE Journal of Solid-State Circuits, vol. 22, no. 5, pp. 841-846, 1987

[13] C.K. Wong, K.K. Lo, P.H.W. Leong, "An FPGA-based Othello Solver" International Conference on Field-Programmable Technology, 2004, pp. 81-88

[14] J. Olivito, R. Gran, J. Resano, C. González, E. Torres, "Performance and Energy Efficiency Analysis of a Reversi Player for FPGAs and General Purpose Processors", Microprocessors and Microsystems, vol. 39, no. 2, pp. 64-73, 2015

[15] T. Watanabe, R. Moriwaki, Y. Yamaji, Y. Kamikubo, Y. Torigai, Y. Nihira, T. Yoza, Y. Ueno, Y. Aoyama, M. Watanabe, " An FPGA Connect6 Solver with a two-stage pipelined evaluation", International Conference on Field-Programmable Technology, 2011, pp. 1-4

[16] T. Yoza, R. Moriwaki, Y. Torigai, Y. Kamikubo, T. Kubota, T. Watanabe, T. Fujimori, H. Ito, M. Seo, K. Akagi, Y. Yamaji, M. Watanabe, "FPGA Blokus Duo Solver using a massively parallel architecture", International Conference on Field-Programmable Technology, 2013, pp. 494-497

[17] K. Koizumi, M. Inaba, K. Hiraki, Y. Ishii, T. Miyoshi, K. Yoshizoe, "Triple Line-Based Playout for Go- An Accelerator for Monte Carlo Go", International Conference on Reconfigurable Computing and FPGAs, 2009, pp. 161-166

[18] M. Boulé, Z. Zilic, "An FPGA Based Move Generator for the Game of Chess", IEEE Custom Integrated Circuits Conference, 2002, pp. 71-74

[19] C. Donninger, A. Kure, U. Lorenz, "Parallel Brutus: the first distributed, FPGA accelerated Chess Program", Proc. 18th International Parallel and Distributed Processing Symposium, 2004

[20] AMBA AXI4 Interface Protocol http://www.xilinx.com/ipcenter/axi4.htm

[21] I. Wu, D. Huang, "A New Family of k-in-a-row Games", Advances in Computer Games, vol. 4250, pp. 180-194

[22] I. Wu, P. Lin, "Relevance-Zone-Oriented Proof Search for Connect6", the IEEE Transactions on Computational Intelligence and AI in Games, vol. 2, no. 3, pp. 191-207, 2010

[23] Vivado Design Suite - HLx Editions. Xilinx, 2016. http://www.xilinx.com/products/design-tools/vivado.html